Reducing Rad-Hard Memories for FPGA Configuration Storage on Space-bound Payloads

Patrick Ostler
Dept. of Electical and Computer Engineering
Brigham Young University
Provo, UT, USA
ospatch@byu.edu

***Abstract*—FPGA use in space-based applications is becoming more common. Radiation-hardened (rad-hard) memories are typically used to store configuration data for programming the FPGA and performing bitstream scrubbing to remove errors in the system that occur from single event upsets (SEUs). Since device densities for the latest FPGAs are growing at a faster rate than rad-hard memories, it is becoming more difficult to reliably store the FPGA configuration data without using a large number of memories. We present a method for cutting the use of rad-hard memories necessary to support the use of the latest FPGA technologies in space-based applications.**

This paper describes a solution to this memory problem which utilizes FPGA partial reconfiguration, with device self-scrubbing, and bitstream compression to create a minimally sized bootstrap design that has a memory footprint that is a fraction of the size of the original design. This bootstrap design is stored locally, and the remaining design elements can be reconfigured as necessary from a remote location. The resulting prototype design yields a compressed bitstream that at less than 2% the size of the bitstream for largest FPGA currently on the market.

I. Introduction

SRAM-based field programmable gate arrays (FPGAs) comprise a unique class of integrated circuits whose functionality can be fully customized after manufacture and tailored to fit a variety of computing needs. For many computing applications, FPGAs offer dramatic performance increases over traditional microprocessor architectures [1] while allowing the flexibility of reconfiguration and a low non-recurring engineering (NRE) cost that is not offered with application-specific integrated circuits (ASICs) [2]. Because of these flexibility and performance advantages, FPGAs are becoming more common on space-bound computing platforms. Examples of current space-computing projects that feature FPGAs include the Mars Rovers [3], Los Alamos National Lab's CFESat [4], and the Single Event Upset Xilinx-Sandia Experiment (SEUXSE) currently aboard the International Space Station [5].

The function of an FPGA is determined by the contents of its configuration memory. This configuration memory is made up an array of static random-access memory (SRAM) cells, and the content of these cells dictates the operation of logic blocks, memories, interconnect, and other modules on the device (see Figure 1 for more detail). The configuration data is programmed to the SRAM of the FPGA on powerup from a bitstream stored on an external source.

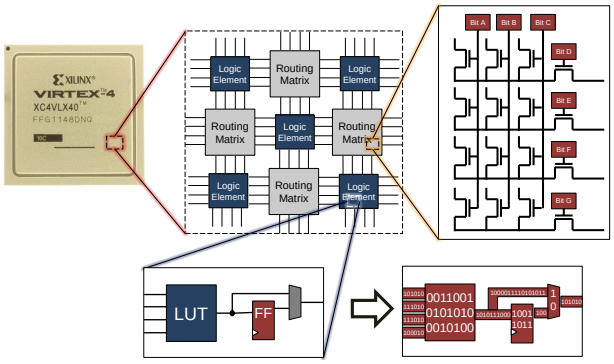

Fig. 1. A basic model of a portion of an FPGA. An FPGA consists of a matrix of logic blocks and interconnect, whose functionality is determined by array of SRAM cells.

Like most digital electronics, FPGAs are susceptible to single-event upsets (SEUs). An SEU is a state changed caused when an ionizing radiation particle strikes a sensitive node on an electronic device. In other words, an SEU can cause the value of a bit in a memory structure to flip. SEU manifestation is a function of the radiation levels and energies in a given environment. SEUs in are rarely observed in electronics at sea level but are encountered frequently in Earth's orbit and beyond.

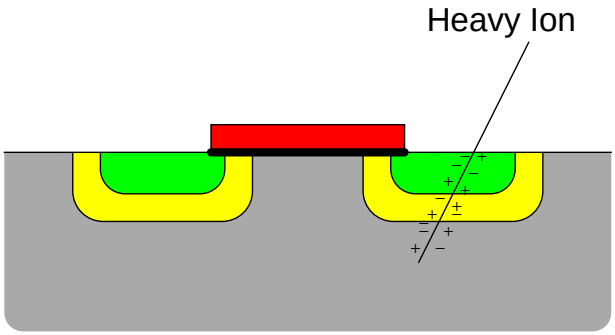

Fig. 2. A diagram of a high-energy heavy ion interacting with a MOSFET. Ionizing radition particles that interact with sensitive nodes on a digital semiconductor device can cause single event upsets (SEUs). [6].

SEUs are a common concern for any electronic device

targeted for a space environment, since data integrity must preserved for computation to occur predictably and accurately. The SEU problem is only compounded for FPGAs, since stored data also accounts for the functionality and operation of the device. As exemplified in Figure 3, an SEU occurence in a routing block can divert signals to an unknown location, and an SEU affecting a logic block can change type of computation performed for a given set of input signals.

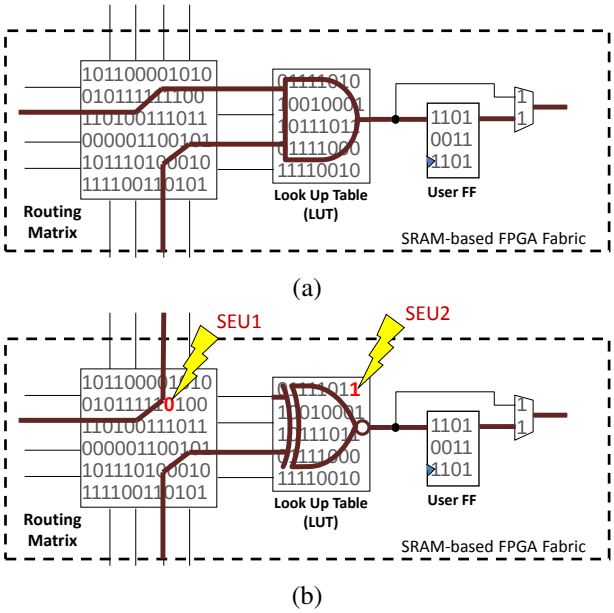

Fig. 3. The block diagram shown in (a) is a simple circuit programmed to a small portion of an FPGA. This circuit is operating in the intended, error-free state. The block diagram in (b) shows the same circuit as seen in (a) after the occurence of an SEU in the routing matrix and an additional SEU in the look-up table. As shown, SEUs can change the intended functionality of design on the device.

For most space-based electronics, mitigation of SEUs is achieved by manufacturing the device to withstand the harsh radiation environment of space. This is often accomplished with proper application of materials, fabrication techniques, and redundancy. Radiation tolerant electronics, however, represent only a very minute portion of the overall electronics market. Since the monetary cost to fabricate a relatively small number of parts for a limited number of interested aerospace firms on the latest state-of-the-art processes is prohibitively high, most companies rely on process methods that are several generations old to manufacture new parts. As a result, the capability and performance of radiation hardened (rad-hard) parts is far beneath those that are typically seen in consumer electronics.

To allow for the use of the latest, commercially available, non rad-hard parts, SEU mitigation methods have been developed to allow for the newest FPGAs to function reliably in all but the most extreme radiation environments. The more commonly used mitigation techniques typically require some type of configuration memory scrubbing. Memory scrubbing is the process of systematically refreshing the memory array with an error-free copy of the data, effectively counteracting the occurence of SEUs. The 'golden' copy of the configuration memory is stored in rad-hard programmable read-only memory (PROM) located adjacent to the FPGA. This memory is not only used to perform scrubbing, but is also used to program the full configuration memory whenever power to the FPGA is cycled.

As device densities continue to grow at a faster rate for commercial devices than for rad-hard parts, it is becoming increasingly difficult to reliably store the FPGA configuration data on a satellite payload. The density of the newest, largest FPGA on the market, the Xilinx Virtex 7 XC7VH870T, is 265.1 Megabits(Mb). For comparison, the rad-hard PROMs currently on the market are only 20Mb in size. In other words, it takes as many as 14 rad-hard PROMs to support just one of the latest FPGAs in space. From a size and power perspective, such a setup can be costly to implement on a satellite payload.

To resolve this issue, we have developed a method to vastly reduce the amount of rad-hard memories necessary to support the latest FPGA technologies in space. This method combines recent developments in module-based partial reconfiguration (PR) design & implementation procedures with known FPGA bitstream compression methods and self error detection & correction (EDAC) techniques. The end result is a minimally sized bitstream, stored on local PROMs, that represents only a fraction of the overall design and is capable of bootstrapping the remainder of the configuration process over a high-speed communication link. The bootstrap design contains only a communication interface, capable of linking with a ground-based station, and the device self-scrubber. The remaining portions of the overall design are later uploaded and programmed as necessary from a ground-based station, and since error correction is done internally, there is no need to refer to a design stored in off-chip memories for scrubbing procedures.

This work documents the means necessary to create and implement a bootstrap FPGA design for reducing rad-hard memory use. Section II will discuss the basics of partial reconfiguration, followed by a description of device self-scrubbing in section III and a short synposis of bitstream compression in IV. The integration of these ideas is brought together in section V, which includes the results of expected bitstream memory reduction when this method is applied.

II. Partial Reconfiguration

Dynamic partial reconfiguration (PR) is the process of modifying an application residing on an FPGA by reprogramming portions of the configuration memory during normal device operation. Most designs programmed to an FPGA are intended to remain static for the life of the application. PR allows for designs to be updated on an FPGA without interrupting the service of critical modules presently on the device. Limited forms of PR have been available for several years [7], [8]. These solutions, however, only allowed for incremental changes to be made to an FPGA design. In recent months, Xilinx has created a tool flow to support the reconfiguration of entire design modules on an FPGA. Module-based PR opens the

possibility to time multiplex several designs within the same region on the FPGA.

Creation of a module-based PR design begins at the HDL level. The HDL design and device resources are partitioned into static and reconfigurable portions, and this partitioning persists through each step of the tools flow. The final result is a static bitstream, used to program the device on powerup, and several reconfiguration bitstreams, each representing a reconfigurable module targeted for the device, that can be programmed to a reconfigurable partition any time after the initial configuration.

The actual reconfiguration process differs slightly from a standard, full FPGA configuration procedure, which halts the operation of the device before and during the programming process. The shutdown process performed for a standard configuration effectively freezes the device when configuration begins until the entire configuration memory array has been programmed and necessary error checks have been performed. PR procedures do not place the device in shutdown mode, allowing critical tasks continue to execute uninterrupted. This degree of freedom makes it possible to place the static and reconfigurable portions of a design into in unknown state as a result of reconfiguration [9]. Therefore, it is necessary to comply with known PR design procedures to ensure proper device reconfiguration.

For this project, PR is used to create a static design using only a minimal amount (under 5%) of logic resources on the FPGA. This design includes:

- A high-speed communication interface – The interface is capable creating and maintaining a communication link with a remote host.
- An ICAP instance – The internal configuration access port (ICAP) is one of the multiple configuration interfaces available on a Xilinx FPGA. The ICAP is different from other configuration interfaces in that it is only accessible only to logic on the FPGA. The communication interface is wired directly to the ICAP to allow for the device to be reconfigured from the remote host.
- Device self-scrubber – This will be described in more detail in the following section.

The remaining resources not allocated for the static region are specified as reconfigurable. In the initial configuration bitstream, the reconfigurable partition (RP) is left blank, so that the bitstream can be compressed as much as possible. The RP can be populated any time after the initial configuration from the remote host at the discretion of the designer.

III. Device Self-Scrubbing

Partial reconfiguration is capable of reducing the size of the initial bitstream needed to program an FPGA, with the idea that most of the intended functionality will be programmed from a remote location sometime after the initial configuration. Most bitstream scrubbing procedures, however, require that the design bitstream, in its entirety, be stored in close proximity to the FPGA allow for prompt detection and correction of errors. This problem is easily rectified by using onboard

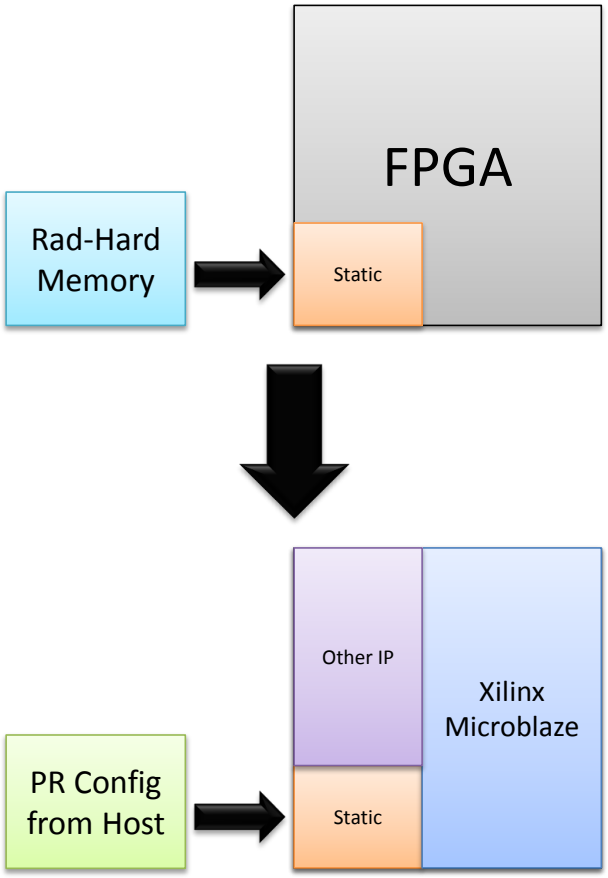

Fig. 4. The idea behind the static region is to essentially ”boot” the FPGA. Once the initial configuration has been performed, the remaining design portions can be programmed later from a remote host.

error detection capability, an addition to the most recent Xilinx Virtex FPGA architectures, with frame readback and reconfiguration procedures.

Device self-scrubbing is made possible on Virtex 4/5/6 FPGA architectures by the use of Hamming single error detect, double error detect (SECDED) codes stored in each configuration frame and two specialized modules integrated into the reconfigurable fabric – the internal configuration access port (ICAP) and the frame error correcting code logic (Frame ECC). Each of these elements is described below:

- SECDED Codes – Configuration frames are comprised of forty-one 32-bit words, or 1,312 bits. Twelve of these bits are reserved to store the ECC. The ECC data for each frame is generated during the bitstream generation process by BitGen[1].
- The ICAP – As described in the previous section, the ICAP is the only configuration interface directly accessible from the logic on the FPGA. It is used to read the

[1]Versions of Xilinx BitGen prior to version 12.1 fail to calculate the ECC data properly on a small number of configuration frames in Virtex 5 applications. The scrubber presented by [10] accounts for this and performs an initialization phase to recalculate incorrect ECC data on design startup prior to activating the scrubber.

configuration frames from memory to detect errors and write corrected frames back to memory.

- The Frame ECC – A circuit that generates syndrome values, based on Hamming SECDED codes located in each frame, for any frame being read from the configuration memory by one of the FPGA configuration interfaces (ICAP, SelectMAP, or JTAG) [11]. A non-zero syndrome generated by the Frame ECC for a configuration frame indicates the presence of one or more errors in the frame. When syndrome bit 11 is asserted, this indicates the presence of a single bit error. Bits 10:0 can be used to determine the location of the bit error. If syndrome bit 11 is not asserted, and syndrome bits 10:0 are non-zero, a double bit error has occured. Because the Hamming codes are SECDED, the presence of more than two bit errors are indeterminable.

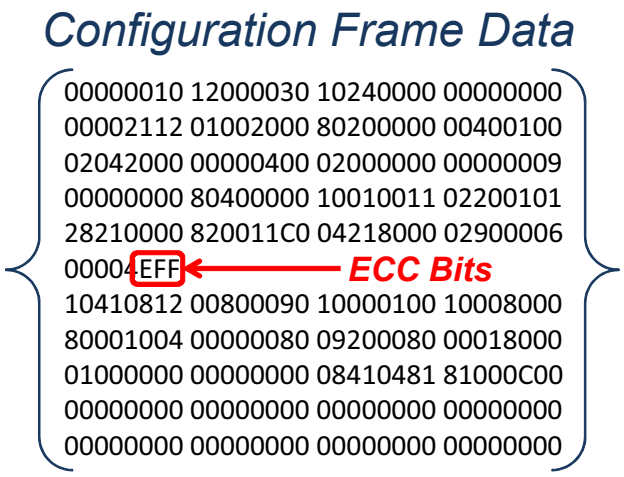

Fig. 5. The configuration frame for Xilinx Virtex 5 device consists of forty-one 32-bit words. Shown here is an example frame, in hexadecimal representation, on a design targeted for a Virtex 5 LX110T. The ECC encompasses 12 bits and is stored in the 21st word. [11].

To self-scrub the configuration memory on an FPGA application, the entire configuration is continually read from memory using the ICAP[2]. If at any time a non-zero syndrome is generated for a frame by the Frame ECC, configuration readback is paused, and the type of error is determined. If a single-bit error is detected, the erroneous frame is read from configuration memory into Block RAM (BRAM) using the ICAP, the syndrome is used to determine the location of the error, the erronous bit is flipped, the frame is written back to the configuration using the ICAP, and readback is continued. If a double-bit error is detected, it is ignored. In this case, scrubbing should be halted, since the configuration is now in an erroneous state that cannot be recovered without external intervention. It is also ideal for the FPGA application to externally signal the presence of a DBE so that necessary action can be taken to handle the error.

[2]The process of continually reading the configuration memory must be accomplished manually by the FPGA application for designs targeting the Virtex 4. In Virtex 5/6 devices, this process is done automatically.

It must be noted that while bitstream scrubbing is capable of correcting bit errors that affect logic and routing portions of the configuration, errors affecting data in memory structures, such as BRAMs, LUTRAMs, SRLs, and flip-flops, cannot be detected or corrected with this method. Additionally, the GLUTMASK must be asserted during bitstream scrubbing to ignore data stored in LUTRAMs and SRLs during readback to insure proper syndrome calculation by the Frame ECC. This limitation is common among all forms of FPGA configuration scrubbing, not just self-scrubbing procedures. These types of errors are usually corrected through some type of design redundancy.

IV. Bitstream Compression

To get the static design bitstream into as small of memory footprint as possible, we compress the bitstream to a fraction of its original size. Numerous techiques currently exist for performing data compression. Many of these methods rely on complex algorithms for compression and decompression. For this project, a compression method that does not require decompression is necessary, since decompression adds an additional level of complexity to the system. Fortunately, there are two methods of compression available that make this possible. The first method is included as an optional feature in the bitstream generation tools from Xilinx. The second technique, developed by Sellers, et. al. [12], was developed for a project similar to this. We opt for the latter, since it yields much better compression ratios on average.

Each of these compression techniques relies on a large amount of unused logic in an FPGA design. Bitstream compression essentially removes the data in the bitstream responsible for controlling portions of the device that go unused by the present design. The result is a bitstream contraining only the configuration data pertinent to the design itself. The steps of the compression process developed by Sellers, et. al., are shown in 6.

V. Prototype Bootstrap Design

To prove the concept of this work as feasible, we created a prototype design to show that bootstrapping and device self-scrubbing from a minimally sized bitstream could be accomplished in practice. This design was targeted for a Virtex-5 LX110T FPGA residing on a Xilinx ML509 (XUP) development board. The design follows the PR tool flow, and the design is broken up into static and reconfigurable portions.

The heart of the static design is the Virtex-5 Soft Error Mitigation (SEM) Core developed at Xilinx [10]. This module is an implementation of the device self-scrubber previously described. In the SEM core, the scrubbing process is executed by an Xilinx Picoblaze microcontroller. To make PR possible, we leveraged the PCI Express capability of the XUP board, and added a PCIe core to fill the role of the communication interface. For our implementation of PR, we normally require the ICAP be wired directly to the communication interface. The ICAP, however, is already instanced and controlled by

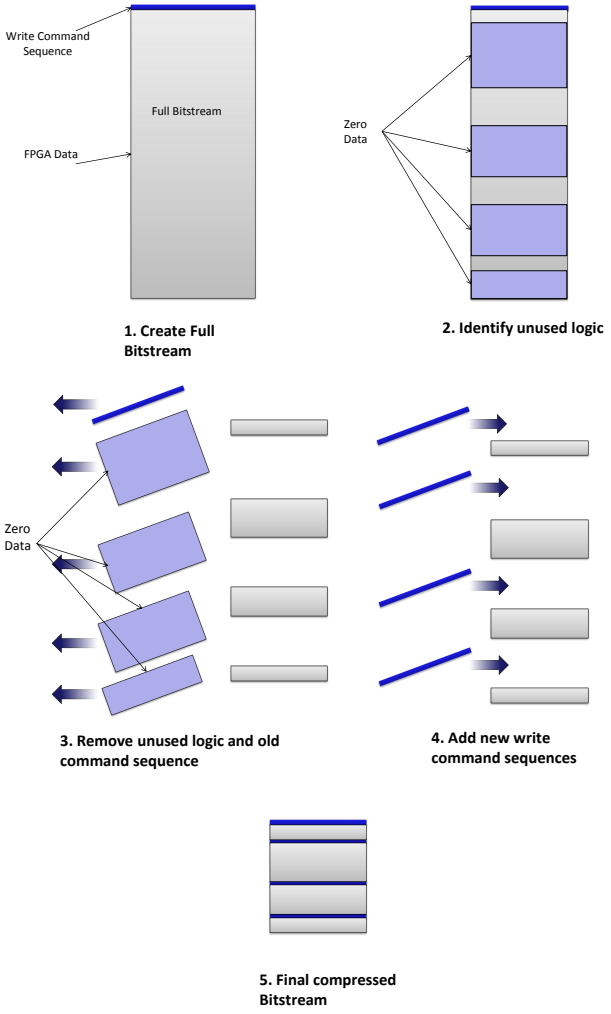

Fig. 6. A diagram of the steps performed by the bitstream compression method developed by Sellers, et. al. [12] Using this technique, empty data frames in the bitstream that serve no function the user design are removed, and new write sequences are added to account for the removed data.

the self-scrubber. Instead of wiring the ICAP to the communication interface, we opted to wire it to the Picoblaze. This allows the host device to interrupt the scrubbing process on the Picoblaze any time the system is in an error free state, and place the system in reconfiguration mode. The host device can then perform reconfiguration and return the system back to scrubbing mode when finished.

For the reconfigurable design, we implemented a Xilinx Microblaze microprocessor core. The Microblaze was chosen because it represents a fairly complex design that is commonly used to create embedded systems on FPGAs. Minor modifications were made to the vanilla Microblaze module created with the Xilinx EDK software. These included removing the capability for debugging software on the Microblaze and moving the clocking interface to the static region of the design. Both of these changes were necessary for PR, as some of the hard IP blocks on the FPGA that are utilized by these elements are not reconfigurable. The software created to run on the Microblaze was designed to print out a message at regularly scheduled intervals to the onboard UART to signal to the host device that reconfigurable design was functional.

The static portion of the overall design utilized a total of 4% of the logic resources on the Virtex-5 LX110T part (2555 out of 69120 total LUTs). Since we must also leave extra resources available for the place and route steps of the design implementation process, we created the static partition on the device to encompass roughly 7% of the logic resources (4800 LUTs). This leaves the remaining 93% to be used by the Microblaze design. The compressed bitstream, generated for the static design with the blank reconfigurable partition is 412KB in size and 11% the size of the original (3799KB). These results are very positive, since we expect the size of the compressed static bitstream to remain constant when implemented on larger FPGAs. At this size, the compressed bootstrap bitstream is only 1.2% of the size of the normal bitstream for a Virtex 7 XC7VH870T FPGA, and the rad-hard memory needed for implementing this FPGA on a space-bound payload could be reduced from fourteen 20Mb PROMs to just one 4Mb PROM.

It should be noted that in this design, we did not account for design failure as a result of an SEU affecting the bitstream scrubber. This problem can easily be fixed with proper application of circuit redundancy [13].

VI. Conclusion

With the densities of rad-hard memories growing at a much slower rate than the densities of FPGAs, it is becoming increasingly difficult to implement the latest FPGA technologies on space-bound computing platforms without using a large number of rad-hard memories. By combining FPGA partial reconfiguration with device self-scrubbing techniques and bitstream compression methods, we have shown how bootstrapping can reduce the amount of rad-hard memories necessary to implement the largest FPGAs on the market for space applications by as much as 98%. This was accomplished by creating an FPGA application capable of performing error correction and reconfiguration over a high speed link. The generated bitstream for the design was then compressed to a small fraction of the original size. In order to make this design fully capable of operating in the space environment, future work is necessary to add design redundancy to mitigate against errors that occur in the device self-scrubbing module.

Acknowledgment

This project was funded by the National Nuclear Security Administration, Office of Nonproliferation Research and Development, NA-22, and the NASA Rocky Mountain Space Grant Consortium.

References

[1] S. Asano, T. Maruyama, and Y. Yamaguchi, "Performance comparison of FPGA, GPU and CPU in image processing," in *Field Programmable Logic and Applications, 2009. FPL 2009. International Conference on*, Sept 2009, pp. 126–131.

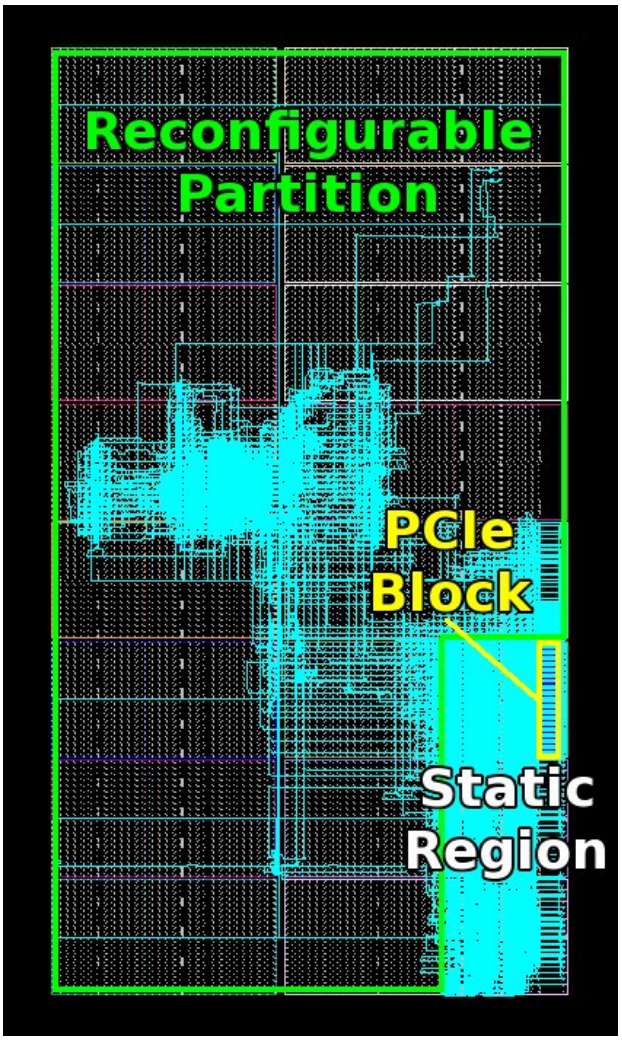

Fig. 7. The post place-and-route design layout for the prototype application on the Xilinx Virtex 5 LX110T part. The static and partial regions have been highlighted to show how the device was partitioned.

[2] D. D. Sherlekar, "Design considerations for regular fabrics," in *Proceedings of the 2004 international symposium on Physical design*, ser. ISPD '04. New York, NY, USA: ACM, 2004, pp. 97–102. [Online]. Available: http://doi.acm.org/10.1145/981066.981087

[3] D. Ratter, "FPGAs on mars," *Xcell*, no. 50, pp. 8–11, 2004.

[4] M. Caffrey, K. Morgan, D. Roussel-Dupre, S. Robinson, A. Nelson, A. Salazar, M. Wirthlin, W. Howes, and D. Richins, "On-orbit flight results from the reconfigurable cibola flight experiment satellite (CFE-Sat)," in *Proceedings of the IEEE Symposium on FPGAs for Custom Computing Machines (FCCM '09)*, K. L. Pocek and J. M. Arnold, Eds., IEEE Computer Society. IEEE Computer Society Press, April 2009.

[5] M. Santarini, "Xilinx boards international space station," *Xcell*, no. 70, p. 4, 2010.

[6] D. E. Johnson, "Estimating the dynamic sensitive cross section of an FPGA design through fault injection," Master's thesis, Brigham Young University, Apr 2005.

[7] *Xilinx 6200 Preliminary Data Sheet*, Xilinx, Inc., San Jose, CA, 1996.

[8] *Two Flows for Partial Reconfiguration: Module Based or Difference Based, XAPP290 (v1.2)*, Xilinx, Inc., September 2004.

[9] *Partial Reconfiguration User Guide, UG702 (v 12.1)*, Xilinx, Inc., May 2010.

[10] K. Chapman, *New Generation Virtex-5 SEU Controller (vA.2)*, Xilinx, Inc., Feb 2010.

[11] *Virtex-5 FPGA Configuration User Guide, UG191 (v3.9.1)*, Xilinx, Inc., Aug 2010.

[12] B. Sellers, J. Heiner, M. Wirthlin, and J. Kalb, "Bitstream compression through frame removal and partial reconfiguration," in *Field Programmable Logic and Applications, 2009. FPL 2009. International Conference on*, Sep 2009, pp. 476–480.

[13] J. Heiner, N. Collins, and M. Wirthlin, "Fault tolerant ICAP controller for high-reliable internal scrubbing," in *Aerospace Conference, 2008 IEEE*, Mar 2008.